

Taverna Workflows: Syntax and Semantics

Daniele Turi

www.cs.man.ac.uk/~dturi

May 23, 2006

Introduction

Taverna is based on a metalanguage for the category of cpos and partial maps. It is a lambda calculus with cartesian products and a strong monad [Mog91] – the free monoid monad [Jac94]. The strength of the monad accounts for one form of iteration in Taverna - the cross product.

1 Syntax

1.1 Types

$$\tau ::= \text{BaseType} \mid !\tau$$

We use σ and τ to range over types.

1.2 Language

Contexts $\Gamma \equiv x_1 : \sigma_1, \dots, x_n : \sigma_n$

Processors Processors are predefined sequents of the form

$$\Gamma \vdash P : \tau$$

The context Γ is a set of typed inputs while the type τ of P is the type of the output. We use product types for multiple outputs.

Workflows are built using these processors (which can be seen as the axioms of our type theory) in conjunction with the following rules (and the standard weakening, contraction and exchange to introduce, remove and swap variables, respectively).

1.2.1 Cut

Function composition:

$$\frac{\Gamma \vdash P : \sigma \quad \Gamma, x : \sigma \vdash Q : \tau}{\Gamma \vdash \text{let } x = P \text{ in } Q : \tau} \quad (1)$$

1.2.2 Product

Pairing:

$$\frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash Q : \tau}{\Gamma \vdash \langle P, Q \rangle : \sigma \times \tau} \quad (2)$$

Projections:

$$\frac{\Gamma \vdash P : \sigma \times \tau}{\Gamma \vdash \text{fst}(P) : \sigma} \quad \frac{\Gamma \vdash P : \sigma \times \tau}{\Gamma \vdash \text{snd}(P) : \tau} \quad (3)$$

1.2.3 Monad

Composition plus strength:

$$\frac{\Gamma \vdash P : !\sigma \quad \Gamma, x : \sigma \vdash Q : \tau}{\Gamma \vdash \text{let } !x = P \text{ in } !Q : !\tau} \quad (4)$$

Unit and multiplication:

$$\frac{\Gamma \vdash P : \tau}{\Gamma \vdash \text{up}(P) : !\tau} \quad \frac{\Gamma \vdash P : !!\tau}{\Gamma \vdash \text{down}(P) : !\tau} \quad (5)$$

1.2.4 Sequential Composition

Sequential composition (control link in Taverna) is syntactic sugar for a let on a weakened variable:

$$P;Q \equiv \text{let } x = P \text{ in } Q \quad (6)$$

where x does not occur neither in P nor in Q .

$$\frac{\Gamma \vdash P : \sigma \quad \Gamma \vdash Q : \tau}{\Gamma \vdash P;Q : \tau} \quad (7)$$

1.2.5 Cross Product

The cross product iteration is also syntactic sugar for double application of the monad (which in turn is double application of its strength):

$$\text{let } x_1 \times x_2 = P_1 \times P_2 \text{ in } !!Q \equiv \text{let } !x_1 = P_1 \text{ in } !(\text{let } !x_2 = P_2 \text{ in } !Q) \quad (8)$$

$$\frac{\Gamma \vdash P_1 : !\sigma_1 \quad \Gamma \vdash P_2 : !\sigma_2 \quad \Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash Q : \tau}{\Gamma \vdash \text{let } x_1 \times x_2 = P_1 \times P_2 \text{ in } !!Q : !!\tau} \quad (9)$$

1.2.6 Dot Product

In contrast with the cross product, the dot product is a primitive.

$$\frac{\Gamma \vdash P_1 : !\sigma_1 \quad \Gamma \vdash P_2 : !\sigma_2 \quad \Gamma, x_1 : \sigma_1, x_2 : \sigma_2 \vdash Q : \tau}{\Gamma \vdash \text{let } x_1 \odot x_2 = P_1 \odot P_2 \text{ in } !Q : !\tau} \quad (10)$$

1.3 Example

As an example let us consider the Scuff workflow `IterationStrategyExample.xml` in Appendix A. Let s denote the base type “text/plain”. The processors are:

- $\vdash \text{Colours} : s$
- $\vdash \text{Animals} : s$
- $\vdash \text{Shapes} : s$
- $x_1 : s \vdash \text{ColoursList} : !s$
- $x_2 : s \vdash \text{AnimalsList} : !s$
- $x_3 : s \vdash \text{ShapesList} : !s$
- $x_4 : s, x_5 : s \vdash \text{ColourAnimals} : s$
- $x_6 : s, x_7 : s \vdash \text{ShapeAnimals} : s$

Then the whole workflow is as follows:

$$\begin{aligned} &\vdash \text{let } x_6 \times x_7 = \\ &\quad (\text{let } x_3 = \text{Shapes in ShapesList}) \\ &\quad \times (\text{let } x_4 \odot x_5 = \\ &\quad \quad (\text{let } x_1 = \text{Colours in ColoursList}) \\ &\quad \quad \odot (\text{let } x_2 = \text{Animals in AnimalsList}) \\ &\quad \quad \text{in !ColourAnimals}) \\ &\quad \text{in !!ShapeAnimals : !s} \end{aligned}$$

2 Operational Semantics

Lazy evaluation [Abr90].

2.1 Rules

2.1.1 Cut

$$\frac{P \Downarrow u \quad Q[u/x] \Downarrow v}{\text{let } x = P \text{ in } Q \Downarrow v} \quad (11)$$

2.1.2 Product

$$\frac{P \Downarrow u \quad Q \Downarrow v}{\langle P, Q \rangle \Downarrow \langle u, v \rangle} \quad (12)$$

$$\frac{P \Downarrow \langle u, v \rangle}{\text{fst}(P) \Downarrow u} \quad \frac{P \Downarrow \langle u, v \rangle}{\text{snd}(P) \Downarrow v} \quad (13)$$

2.1.3 Monad

$$\frac{P \Downarrow \vec{u} \quad \{Q[u_i/x] \Downarrow v_i\}_{i=1..|\vec{u}|}}{\text{let } !x = P \text{ in } !Q \Downarrow \vec{v}} \quad (14)$$

$$\frac{P \Downarrow v}{\text{up}(P) \Downarrow [v]} \quad \frac{P \Downarrow [[w_{11}, \dots, w_{1m}], \dots, [w_{n1}, \dots, w_{nm}]]}{\text{down}(P) \Downarrow [w_{11}, \dots, w_{ij}, \dots, w_{nm}]} \quad (15)$$

2.1.4 Dot Product

$$\frac{P_1 \Downarrow \vec{u} \quad P_2 \Downarrow \vec{v} \quad |\vec{u}| = |\vec{v}| \quad \{Q[u_i/x_1, v_i/x_2] \Downarrow w_i\}_{i=1..|\vec{u}|}}{\text{let } x_1 \odot x_2 = P_1 \odot P_2 \text{ in } !Q \Downarrow \vec{w}} \quad (16)$$

2.1.5 Cross Product

Definition (8) and rule (14) imply the following:

$$\frac{P_1 \Downarrow \vec{u} \quad P_2 \Downarrow \vec{v} \quad \{Q[u_i/x_1][v_j/x_2] \Downarrow w_{ij}\}_{j=1..m}^{i=1..n} \quad |\vec{u}| = n \quad |\vec{v}| = m}{\text{let } x_1 \times x_2 = P_1 \times P_2 \text{ in } !!Q \Downarrow [[w_{11}, \dots, w_{1m}], \dots, [w_{n1}, \dots, w_{nm}]]} \quad (17)$$

2.2 Example (continued)

The operational semantics of the example workflow in §1.3 is as follows.

- Colours \Downarrow “red, green”
- Animals \Downarrow “cat, rabbit”
- Shapes \Downarrow “square, circular, triangular”
- $\text{let } x_1 = \text{Colours in ColoursList} \Downarrow$ [“red”, “green”]
- $\text{let } x_2 = \text{Animals in AnimalsList} \Downarrow$ [“cat”, “rabbit”]
- $\text{let } x_3 = \text{Shapes in ShapesList} \Downarrow$ [“square”, “circular”, “triangular”]
- $(\text{let } x_1 = \text{Colours in ColoursList}) \odot (\text{let } x_2 = \text{Animals in AnimalsList}) \Downarrow$ [\langle “red”, “cat” \rangle , \langle “green”, “rabbit” \rangle]
- $\text{let } x_4 \odot x_5 = [\langle$ “red”, “cat” \rangle , \langle “green”, “rabbit” $\rangle]$ in $! \text{ColourAnimals} \Downarrow$ [“red cat”, “green rabbit”]
- $x_6 \times x_7 = [[\langle$ “square”, “red cat” \rangle , \langle “square”, “green rabbit” $\rangle]$,
 $[\langle$ “circular”, “red cat” \rangle , \langle “circular”, “green rabbit” $\rangle]$,
 $[\langle$ “triangular”, “red cat” \rangle , \langle “triangular”, “green rabbit” $\rangle]]$

Then:

```
let x6 × x7 =
  (let x3 = Shapes in ShapesList)
  × (let x4 ⊙ x5 =
     (let x1 = Colours in ColoursList)
     ⊙ (let x2 = Animals in AnimalsList)
     in !ColourAnimals)
  in !!ShapeAnimals ↓
[[“square red cat”, “square green rabbit”],
 [“circular red cat”, “circular green rabbit”],
 [“triangular red cat”, “triangular green rabbit”]]
```

3 Observational Equivalence

Value-passing bisimulation [FT01].

4 Denotational Semantics

Interpret types as cpos, the monad constructor as the (continuous) free monoid monad, and workflows as partial continuous functions [Plo85].

For simplicity, let us consider the free monoid monad T in the category of sets. Its unit $\text{up} : A \rightarrow TA$ maps an element a of a set A to the one element list $[a]$. Its multiplication $\text{down} : T^2A \rightarrow TA$ flattens a list of list into a single list. There is also a unique strength operation

$$t : A \times TB \rightarrow T(A \times B)$$

which maps an element a of A and a list $[b_1, \dots, b_n]$ of elements of B to the list of pairs $[\langle a, b_1 \rangle, \dots, \langle a, b_n \rangle]$.

5 Future Work

- Define the right notion(s) of observational equivalence.
- Model the dot product categorically.

References

- [Abr90] S. Abramsky. The lazy lambda calculus. In D. A. Turner, editor, *Research Topics in Functional Programming*, pages 65–116. Addison-Welsey, Reading, MA, 1990.

- [FT01] Marcelo Fiore and Daniele Turi. Semantics of name and value passing. In *Proc. 16th LICS Conf.*, pages 93–104. IEEE, Computer Society Press, 2001.
- [Jac94] Bart Jacobs. Semantics of weakening and contraction. *Ann. Pure Appl. Logic*, 69(1):73–106, 1994.
- [Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.
- [Plo85] Gordon Plotkin. Lectures on predomains and partial functions. Notes for a course at CSLI, Stanford University, 1985.

Appendix A

IterationStrategyExample Workflow

```
<?xml version="1.0" encoding="UTF-8"?>
<s:scufl xmlns:s="http://org.embl.ebi.escience/xscufl/0.1alpha"
  version="0.2" log="0">
  <s:workflowdescription
    lsid="urn:lsid:www.mygrid.org.uk:operation:VI9FMF5HBQ10"
    author="Tom Oinn"
    title="Demonstration of configurable iteration">
    This workflow shows the use of the iteration strategy editor
    to ensure that only relevant combinations of inputs are used
    during an implicit iteration.
  </s:workflowdescription>
  <s:processor name="Colours">
    <s:description>
    Contains the list of colours to be used later in the workflow
    </s:description>
    <s:stringconstant>red, green</s:stringconstant>
  </s:processor>
  <s:processor name="ShapeAnimals">
    <s:description>
    This concatenation doesn't have an explicit iteration strategy
    so will use the default; this means that the process will be invoked
    over all possible combinations of its inputs.
    For an input set of three items in 'string2' and two in 'string1'
    this will therefore iterate six times and produce an output list
    of length six.
    </s:description>
    <s:local>org.embl.ebi.escience.scuflworkers.java.StringConcat</s:local>
  </s:processor>
  <s:processor name="ColourAnimals">
```

```

<s:description>
  Create a description of a coloured animal. In this case we're using
  an explicit iteration strategy declaration to state that the inputs
  should be treated as linked; rather than iterating over all combinations
  of 'string1' and 'string2' we combine the nth elements of each one.
  This of course only works if the iterators have the same size.
</s:description>
<s:local>org.embl.ebi.escience.scuflworkers.java.StringConcat</s:local>
<s:iterationstrategy>
  <i:dot xmlns:i="http://org.embl.ebi.escience/xscufliteration/0.1beta10">
    <i:iterator name="string2" />
    <i:iterator name="string1" />
  </i:dot>
</s:iterationstrategy>
</s:processor>
<s:processor name="ShapesList">
  <s:description>
    Splits the shapes string value into a list, as no regex is supplied
    it uses the default ','
  </s:description>
  <s:local>org.embl.ebi.escience.scuflworkers.java.SplitByRegex</s:local>
</s:processor>
<s:processor name="AnimalsList">
  <s:description>
    Splits the animals string value into a list,
    as no regex is supplied it uses the default ','
  </s:description>
  <s:local>org.embl.ebi.escience.scuflworkers.java.SplitByRegex</s:local>
</s:processor>
<s:processor name="ColoursList">
  <s:description>
    Splits the colours string value into a list, as no regex is supplied
    it uses the default ','
  </s:description>
  <s:local>org.embl.ebi.escience.scuflworkers.java.SplitByRegex</s:local>
</s:processor>
<s:processor name="Shapes">
  <s:description>
    Contains a list of shapes to be used later in the workflow
  </s:description>
  <s:stringconstant>square, circular ,triangular</s:stringconstant>
</s:processor>
<s:processor name="Animals">
  <s:description>
    Contains a list of animals to be used later in the workflow
  </s:description>

```

```
    <s:stringconstant>cat, rabbit</s:stringconstant>
</s:processor>
<s:link source="Shapes:value" sink="ShapesList:string" />
<s:link source="Animals:value" sink="AnimalsList:string" />
<s:link source="Colours:value" sink="ColoursList:string" />
<s:link source="ColoursList:split" sink="ColourAnimals:string1" />
<s:link source="AnimalsList:split" sink="ColourAnimals:string2" />
<s:link source="ColourAnimals:output" sink="ShapeAnimals:string2" />
<s:link source="ShapesList:split" sink="ShapeAnimals:string1" />
<s:link source="ShapeAnimals:output" sink="Output" />
<s:sink name="Output" />
</s:scufl>
```